Big Data Processing Systems Final Project

Gonçalo Matos
Faculdade de Ciências e Tecnologia
Faculdade Nova de Lisboa
Email: gs.matos@campus.fct.unl.pt

Gonçalo Traça
Faculdade de Ciências e Tecnologia
Faculdade Nova de Lisboa
Email: g.traca@campus.fct.unl.pt

Sofia Morgado
Faculdade de Ciências e Tecnologia
Faculdade Nova de Lisboa
Email: sv.morgado@campus.fct.unl.pt

***Abstract*—Analysis of large datasets is being performed at an unprecedented frequency. Several technologies have been developed to do so, offering a variety of solutions and drawbacks related to the processing of different data types and data processing requirements. Here, we compared the performance of five different technologies – MapReduce, Spark RDD, SparkDF, Spark SQL and Hive – to address five questions related with air pollution in the USA. We have found that Spark technologies have the best performance times. We identified some caveats which include the time-consuming implementation of MapReduce or the increased performance time of Hive. Altogether, for this dataset we consider Spark technologies the most accessible to implement, especially SparkDF and SQL, that analyze structured data.**

1. Introduction

Due to the technological advances of the last decades, large and hard-to-manage datasets – referred to as Big Data - are being generated every day and used in virtually all areas of human society, including scientific research, financial markets, health or sports [1]. This motivated the development of new storage and processing tools which are conceptually innovative and provide an array of solutions to efficiently analyze and add value to big data [2].

The first of such tools is Hadoop MapReduce (MR). MR programs are sequences of Map tasks, that generate intermediate key/value pairs, and Reduce tasks, which merge values associated with each intermediate keys. These programs perform batch processing on unstructured data and are automatically paralleled, distributed across several commodity machines, and fault tolerant, which is guaranteed by data replication. However, replication, combined with the need to send data across the network and for it to be written in the disk between tasks, slows down the process. Moreover, program reuse may be limited [3].

Apache Spark offers fault-tolerance, performs in-memory computations in large clusters and offers high-level abstraction. Spark relies on Resilient Distributed Datasets (RDDs), immutable distributed collections of objects, which are built from others through lazy parallel transformations. Fault tolerance is ensured as RDDs contain information on how they were derived, allowing recalculation. Despite this, Spark relies on big memory usage and is not able to deal with structured datasets [4].

Several tools allow working with structured datasets, including the Spark SQL and Spark DataFrame (Spark DF) APIs and Apache Hive. DataFrames API introduced the usage of structured datasets, with named columns that can be manipulated using Spark functionalities. These provide easier solutions to work with than typical Spark, benefiting from going through the Catalyst optimizer. Spark SQL integrates the SQL functional programming with relational processing. It includes the Catalyst query optimizer that optimizes structural queries, speeding computation [4], [5]).

Finally, Hive is an open-source distributed data warehousing database which operates on Hadoop Distributed File System (HDFS), from Hadoop. It allows users to query and analyze big data using a SQL-based interface called HQL, from data stored in tables. HQL facilitates the usage of MR programs, providing easier programming solutions [6].

For this project, we used these technologies to answer several questions related with air pollution in the United States of America (USA). By tackling the same question with different methodologies, we were able to understand which strategies were more efficient and easier to implement in the context of this dataset.

2. Methods

To answer the five proposed questions, information was collected from given datasets and five different technologies were implemented for each question.

2.1. Datasets

In this project, information about air pollution in the USA was extracted from the data file epa_hap_daily_summary-small.csv, with 117MB. This dataset contains data for every monitor in the Environmental Protection Agency (EPA) database for each day. Each entry contains a daily summary record that either contains the aggregation of all sub-daily measurements taken at the monitor or a single sample value, if the monitor takes a single, daily sample. For each question, the header and empty lines were removed and different fields of interest were selected.

For MR and Spark, supplementary data in the usa_states.csv document was used to create a dictionary, "coordinates", with the information about the limit coordinates of each state. Also, the mean of the latitude and longitude of each state was calculated to find the center and quadrants of the states, and added to the dictionary. For the technologies using structured data, a suplemmental dataframe was created with these informations. This data was used in the last two questions.

2.2. Technologies implemented

For MR, solutions using sequences of map and reduce functions were implemented. Firstly, using the module sys, the mapper received the csv dataset as input. Then each line of the data was split, and the fields of interest were selected by their index. The output of one reducer was either used as input to the next MR or was the final solution for the question.

A SparkContext was created to start Apache Spark functionality. For plain Spark, the data was imported using the function textFile. After the relevant components of each RDD were collected using a map function, multiple other functions were applied to obtain the output. The data was also imported using the textFile function for Spark DF, whereas in this case the data of interest was structured into Dataframes for further data manipulation. For SQL, the function SQLContext was used to implicitly convert an RDD to a DataFrame and the data was imported using load function.

Finally, for hive, the function %%file was used to import data. Similarly, a sequence of functions was applied for each question.

In the next chapter, we discuss the specific implementation for each of the proposed questions and the results obtained.

3. Results

3.1. Which states have more/less monitors?

We reasoned that each monitor should have a unique location, defined by its latitude and longitude. Unique combinations of these coordinates were searched and used to count how many different monitors existed by state. This methodology was also used in the last two questions.

For MR, this process required a MR cycle, where the output of the Map function included a key, the latitude and longitude of the machine, and a value, the state name; this pair was fed to the reduce function, already sorted, and repeated machines were discarded. For spark, a ReduceByKey function was used. For Spark DF, Spark SQL and Hive, COUNT DISTINCT was used. This created a new column already containing the count of machines per state. The number of unique monitors was then count for each state for the remaining technologies and the results were sorted from the highest to the lowest number of monitors. For MR, this was achieved using two MR cycles, one for each task. For spark, simply a ReduceByKey(sum) was followed by a sortBy function. For Spark DF, Spark SQL and Hive, the results were sorted using sort or ORDER BY functions.

TABLE 1. NUMBER OF MONITORS FOR THE TWO STATES WITH THE LARGEST AND SMALLEST VALUES

State	Total
California	170
Texas	133
...	...
Hawaii	6
District of Columbia	5

We concluded that California and Texas had the highest number of monitors, whereas the District of Columbia and Hawaii had the fewest (Table 1).

3.2. Which counties have best/worst air quality?

To evaluate air quality per county, we calculated the mean value of the arithmetic mean for the pollutants, for each county.

Two MR cycles were implemented, one for computing the mean of the registered arithmetic mean of pollutants by dividing the sum of each measurement to the count of measurements, and the other for sorting the results. The same idea was computed using Spark mapValues to add the count, a reduceByKey to sum the values measured and the measurements, a mapValues function to divide them and a sort to order them.

With Spark DF, a groupBy("county") followed by an aggregating function were used to calculate the mean of pollutant arithmetic mean for each county. The results were ordered by descending air quality using sort.

For Spark SQL and Hive, a simple query using the function GROUP BY county name was applied after the mean pollution was calculated.

Counties with worst air quality were Tipton and Nassau, those with the best air quality were Sweet Grass and Martin (Table 2).

TABLE 2. MEAN VALUES OF THE ARITHMETIC MEAN PER COUNTY WERE CALCULATED AS A PROXY FOR AIR QUALITY. DATA FOR THE TOW STATES WITH HIGHER AND LOWER VALUES IS DISPLAYED

County	Pollutants
Tipton	2556.000
Nassau	19.000
...	...
Sweet Grass	0.000
Martin	0.000

3.3. Which states have best/worst air quality in each year?

For this exercise, the mean of the arithmetic mean of the pollutants level was calculated for each combination of

TABLE 3. MEAN VALUES OF THE ARITHMETIC MEAN PER STATE AND PER YEAR WERE CALCULATED AS A PROXY FOR AIR QUALITY. DATA FOR THE TWO COMBINATIONS WITH HIGHER AND LOWER VALUES IS DISPLAYED

State	Year	Pollutants
Tennessee	1990	170.401
Country of Mexico	1995	8.460
...	...	...
Wisconsin	1990	0.000
Ohio	1991	0.000

year and state, and the results were sorted with a descending order. For MR, the output of the map was a key containing the name of the state and the information of the year, obtained from the date local column of the original dataset, and value contained the values for the arithmetic mean. The reducer produced a tuple containing the same key as the mapper but the value was obtained by the sum of all measurements done divided by the number of measurements taken, on a given state and in a given year. For spark, we used map to generate the tuple containing state and year as key and corresponding arithmetic means as values. Then a first mapValues was used to attribute the value of 1 to each tuple. A reduceByKey function was employed to sum all arithmetic means and count the number of measurements per state and year. Finally, another mapReduce divided the sum of the arithmetic means by the number of measurements and results were sorted using sortBy.In SparkDF, SparkSQL and Hive, the same tuple was obtained using select and groupBy year and state. Final results were obtained using the sum of the arithmentic means divided by the count of arithmetic means gathered by state and year and ordered by descending values..

As shown in table 3, Tennessee state, in the year of 1990, had the worst air quality registered. We found several examples of states that on a given year had pollution values of 0.

3.4. For each state, which is the average distance of the monitors to the state center?

For this question, a ”coordinates” dictionary was used to compare the coordinates of each unique monitor to the center of the state. Only the monitors present in states mentioned in the ”coordinates” dictionary were considered. Monitor distance to the center, for latitude and longitude, was calculated using decimal degrees. Values were transformed to kilometers (111 km to 1 degree) and the Pythagorean theorem was used, to obtain the monitor distance to the center.

After selecting unique monitors, another MR cycle was used to select the states in the dictionary and calculate the mean distance to each state. Since the state was the key of the output of the mapper, the results were already sorted alphabetically by state.

In Spark, filter was used to select the the states in the dictionary and a map was used to calculate the difference in latitude and longitude from the monitor to the city center. A reduceByKey calculated the mean of the module of this differences of monitors from the same state. A sequence of map functions were used to transform the values from degrees to kilometers and to compute the Pythagorean theorem to calculate the distance.

TABLE 4. MEAN DISTANCE IN KILOMETERS OF THE MONITORS POSITION TO THE STATE CENTER, ORDERED ALPHABETICALLY

State	Mean distance
Alabama	167.578
Alaska	637.389
...	...

In SparkDF, two dataframes were created. One contained information from the usa_states.csv file and a map function was used to obtain the min, average and max latitude and longitude for each state. This RDD was then transformed into a dataframe using createDataFrame. To this dataframe two new columns were added using withColumn function that had average latitute and longitude for each state. A second dataframe was built from the main file with the monitor information and a map function was employed to obtain the latitude and longitude of every entry of any state. This RDD was then transformed into a dataframe using createDataFrame. A third dataframe was obtained by joining the two previously mentioned. A series of operations were performed on this dataframe using the withColumn function: i) the difference between the latitude position of a monitor and the latitude of the state center, ii) the difference between the longitude position of a monitor and the longitude, iii) these values were used to calculate the monitor distance to the state center using Pythagoras theorem and iv) were transformed from decimal degrees to kms by multiplying by 111. The values in km were groupBy state and the average distance was calculated using aggregate and sorted by state name. Simmilar approaches were taken for SparkSQL and Hive using the specific functions of these tecnologies.

Results were registered alphabetically (Table 4).

3.5. How many monitors are there per state quadrant?

We defined four quadrants per state, defined by the minimum, maximum and average latitude and longitude coordinates. Using the position of each unique monitor, we placed them in one of the four quadrants and the results were registered (Table 5).

After selecting unique monitors, another MR cycle was used to select the states in the dictionary and from each quadrant.

In Spark, filter was used to select the the states in the dictionary and a filter was used to place each machine in the corresponding quadrant. A map function added the name of the quadrant to the state name for each line in the RDD for each quadrant, and a union function was used to join the

4 RDDs. Finally, a reduceByKey calculated the sum of the machines per quadrant of each state.

In SparkDF, two dataframes identical to those of Q4 were created. A second dataframe was built from the epa file with the monitor information and a map function was employed to obtain the latitude and longitude and the combination of both coordinates of every entry of any state. This RDD was then transformed into a dataframe using createDataFrame. groupBy State was used and unique coordinate combinations were obtained by state using agg(count). To identify the number of monitors per quadrant, four new dataframes, one per quadrant, were built from the previous two using join. To get those monitors found in the NE quadrant only monitors with longitude and latitude above the mean for given state were included in the NE dataframe. SW dataframe contained monitors with both latitude and longitude below the average of given state. NW had latitude values above and longitude below the state average coordinates. SE had latitude values below and longitude above the state average coordinates. The total number of monitors per quadrant was obtained by counting the unique coordinates of each state in each quadrant using agg(countDistinct). The same logic was applied for SparkSQL and Hive, using the equivalent functions in SQL.

TABLE 5. NUMBER OF MONITORS PER STATE QUADRANT, ORDERED ALPHABETICALLY

State	NE	NW	SE	SW
Alabama	5	14	5	7
Alaska	4	4	2	3
...	...	...	...	...

4. Performance times

To measure the performance times we choose to use the Real Time in seconds, which consists of the amount of time it takes for all the execution, including waiting time.

Spark technologies had faster execution times than MapReduce and Hive (Table 6), even in the context of a fairly small dataset (117 MB). As Spark performs in-memory processing and stores less data in disk, its computation is faster than MapReduce or Hive which read and write data to the disk.

Regarding the Spark technologies, SparkSQL was slightly faster than SparkRDD. SparkSQL has a Catalyst optimizer that improves performance and this may explain those differences observed. If this is the case, we expect higher differences in bigger data-sets where the optimization by the Catalyst may be more relevant. We expected SparkDF to be faster than Spark RDD and as fast as SparkSQL. Nevertheless SparkDF was the slowest of the Spark technologies.

Hive was slower than MapReduce although both run on top of Hadoop. Hive technology was built to deal with very large data-sets and not to analyze small data-sets, such as the one used here. This impaired capacity may explain a slower execution time in comparison to MapReduce. Also, we used the ORDER BY function, not always taking advantage of Hive inherent capability of sorting data by key, which may have led to poorer performances.

TABLE 6. PERFORMANCE TIME IN SECONDS FOR EACH TECHNOLOGY AND QUESTION

	MR	Spark	Spark DF	Spark-SQL	Hive
Q1	9.25	2.75	4.40	2.69	13.14
Q2	6.33	2.60	3.93	2.50	13.74
Q3	7.33	2.75	4.00	2.30	19.41
Q4	7.34	2.63	6.03	2.29	17.80
Q5	7.30	3.34	7.02	3.02	132.34

5. Implementation

Procedural language based programming with low level of abstraction may be long and repetitive, whereas Spark implementation require specific functions knowledge. Therefore preference may depend on the user. We consider that MapReduce was more difficult to implement than Spark technologies. The same logical operation that is performed with few lines of code in Spark may require several MapReduce cycles, making it more time consuming.

Among the Spark technologies, having structured data facilitated the program execution and we consider SparkSQL implementation more straightforward than that of Spark DF. Finally, as Hive implementation was similar to that of Spark SQL, we implemented it with relative ease.

6. Conclusion

In this project, we implemented 5 technologies to analyze Big Data to answer 5 questions about air pollution in the USA. The time consuming implementation for MR and or the long performance time of Hive were the main disadvantages found. Spark technologies were easier to implement, especially when structured data was used.

Acknowledgments

We thank professor João Lourenço for useful feedback and availability to respond to our questions. We thank the piazza community and the students of the MAEBD for insightful discussions and sharing information.

Group Members Contribution

The final individual contribution was 33.3% of the project. All members contributed to design a strategy to address the five questions here asked. All members wrote part of the code that obtained the results and contributed for the writing of the manuscript. All members participated in the discussion and interpretation of the data here presented.

References

[1] https://www.sas.com/en_us/insights/big-data/what-is-big-data.html

[2] Buhl, Hans Ulrich, et al. *Big Data* Wirtschaftsinformatik, 2013.

[3] Jeffrey Dean, Sanjay Ghemawat. *MapReduce: simplified data processing on large clusters* Proc. OSDI'04, 2004.

[4] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das et al. *Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing* . In Proc. NSDI'12., 2012.

[5] https://spark.apache.org/docs/latest/sql-programming-guide.html

[6] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain et al. *Hive: a warehousing solution over a map-reduce framework.* Proc. VLDB Endow, 2009.